HW 9

PWM Controller VHDL

EELE 467

Due date: 11/08/2024

A Pulse Width Modulated (PWM) signal is shown below with varying pulse widths. You will be creating a PWM controller that will be used in your final project to control an RGB LED.

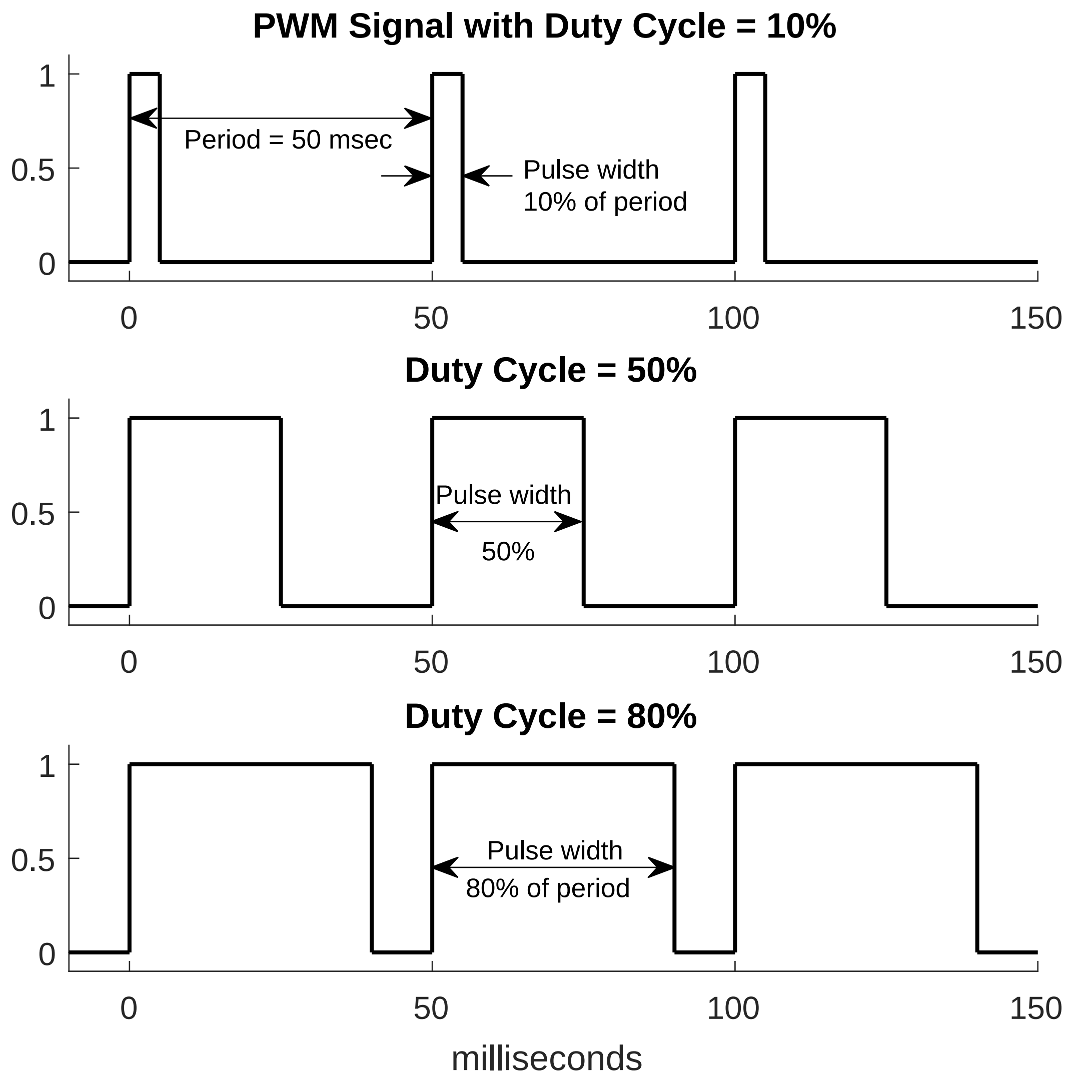

Figure 1: A PWM signal with three different pulse widths (also known as the duty cycle). The top figure has a duty cycle of %10. The middle figure has a duty cycle of %50. The bottom figure has a duty cycle of %80. In all three cases, the period of the pulses is 50 milliseconds.

Your PWM controller will have the following entity (you can use generics for your data types, if you wish):

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity pwm_controller is
  generic (
    CLK_PERIOD : time := 20 ns
  );
  port (
    clk        : in    std_logic;
    rst        : in    std_logic;
    -- PWM repetition period in milliseconds;
    -- datatype (W.F) is individually assigned
    period     : in    unsigned(W_PERIOD - 1 downto 0);
    -- PWM duty cycle between [0 1]; out-of-range values are hard-limited
    -- datatype (W.F) is individually assigned
    duty_cycle : in    unsigned(W_DUTY_CYCLE - 1 downto 0);
    output     : out   std_logic
  );
end entity pwm_controller;
```

Listing 1: PWM controller entity. `W_PERIOD` and `W_DUTY_CYCLE` are assigned in Table 1; you may use generics for those if you wish.

Period The input control signal **period** specifies the repetition period of the PWM square wave in *milliseconds*. The period to be created has been individually assigned in the **PWM Period** column in Table 1, which has units of milliseconds. The **data type** of the period control signal has also been individually assigned in the **period** column where W is the size of the signal and F is the number of fractional bits in the signal. You will need to assume this data type for your design and for your final project.

Duty cycle The input control signal **duty_cycle** specifies the duty cycle of the PWM square wave and is typically given as a *percentage*, which can range from 0 to 100. However, for our purposes, the duty_cycle value will range from 0 to 1 since this will make the fixed-point math easier. The **data type** of the duty_cycle control signal has been individually assigned in the **duty_cycle** column below where W is the size of the signal and F is the number of fractional bits in the signal. Values greater than 1 should be set to 1, and values less than 0 should be set to 0.

Write the PWM controller

Write your PWM controller in `hdl/pwm/pwm_controller.vhd`

Test the PWM controller

Testbench

Write a testbench for your PWM controller:

1. The testbench does not have to be self-checking
2. Test multiple period and duty cycle values
3. You may want to set the `CLK_PERIOD` generic to something larger than 20 ns to speed up your simulation time.

Physical test

We're going to test our PWM controller by instantiating it in the FPGA fabric and mapping the PWM output to a GPIO pin. You'll verify your PWM controller is working by looking at PWM output on an oscilloscope.

Quartus project setup

For your Quartus project, you have two primary options:

- Create a new project based upon your LED patterns project.
- Use your LED patterns project directly.

Since the PWM controller has nothing to do with the LED patterns project, I suggest creating a new project:

1. Move the LED patterns Quartus project files to a subdirectory (e.g., `quartus/led-patterns`)
2. Copy your `quartus/led-patterns` folder and name it something like `quartus/pwm` or `quartus/rgb-led`

In your Quartus project, instantiate your PWM controller in your top-level file.

Connect the output of your PWM controller to a GPIO pin. Refer to the DE10 Nano's manual to see the GPIO pin locations.

Oscilloscope test

Check your PWM output using an oscilloscope. I suggest using the scopes in Cobleigh 601.

You will hardcode period and duty cycle values of your choosing.

Measure the PWM period and duty cycle on the oscilloscope using either cursors or the measurement functions. Take an oscilloscope screenshot that shows the PWM waveform and measurements.

Table 1: Assigned Periods and Data Types. Data types are listed as W.F

Last Name	First Name	period	duty_cycle
Allick	Kristoffer	18.12	12.11
Almnaiee	Jasem	9.3	14.13
Binfet	Caleb	30.24	19.18
Buckley	Peter	31.25	12.11
Calvin	Jessica	20.14	28.27
Crittenden	Ian	20.14	12.11
Culwell	Joshua	16.10	12.11
Currie	Drew	30.24	11.10
Dupuis	Ryan	17.11	12.11
Gill	Nicholas	10.4	18.17
Girardot	Colter	27.21	15.14
Graham	James	17.11	31.30
Guentherman	Zachary	14.8	18.17
Hexom	Jordy	18.12	11.10
Holmes	Riley	10.4	30.29
Howard	Seth	11.5	32.31
Hughes	Jonathon	31.25	18.17
Jensen	David	31.25	9.8
Jones	Kaleb	22.16	13.12
Kaiser	Dirk	9.3	17.16
Kirkland	Grant	13.7	22.21
Lewis	Zane	16.10	13.12
McLean	Aaron	28.22	22.21
Netz	Noah	8.2	25.24
Osborne	Emmett	9.3	12.11
Raber	Dylan	12.6	10.9
Schwartz	Emily	24.18	14.13
Vincent	Kenneth	26.20	15.14
Wilcox	Joshua	24.18	18.17
Wurden	Nicholas	19.13	20.19

Deliverables

Submit your assignment using the workflow that's detailed at `docs/workflow.md` and the submission template at `docs/submission-template.md`.

In your markdown file:

- Include screenshots of your testbench simulation.
- Include your screenshot that shows the PWM signal with the period and duty_cycle of your choosing (document what the period and duty_cycle is in the picture and show the times marked in the oscilloscope window).